universidade de aveiro

Copy Model: Understanding Compression

Teoria Algoritmica da Informação - 40752

Gonçalo Marques - 98648
João Reis - 115513
Renan Ferreira - 93168

Aveiro
March, 2024

Table of Contents

Abstract **3**
Introduction **4**
Methodology 5
    Mutator 5
    Copy Model (CPM) 5
Discussion **6**
    Testing 7
    Results 7
Conclusions **9**
Future Work **10**
References **11**

Abstract

This paper presents a detailed explanation of the development and testing of a copy model algorithm. It dives slightly into the development of a mutator program as well which is used for testing purposes. The copy model builds a probability distribution and estimates the number of bits required to represent symbols in a sequence and therefore makes an estimation for the possible theoretical size of a file upon compression. These values are then compared to general-purpose compression tools.

Introduction

In the landscape of data compression, the copy model is an essential and effective strategy. It works by identifying similarities within a file by predicting what symbol is coming in the future based on a probability distribution measured by what was processed in the past and estimating the number of bits necessary for representing each symbol. Its effectiveness can, however, be affected by the occurrence of events that haven't occurred before. A way to deal with this is to have a model which is adaptive and turns itself off when necessary by measuring the number of hits and misses of its predictions.

This paper focuses on two key objectives: the development of a copy model algorithm and a mutation algorithm.

- The copy model will theoretically evaluate the performance of a compression algorithm with the implemented model. The copy model is meant to build a probability distribution and estimate how many bits are needed to represent each symbol individually as well as the average bits needed to represent the entire file, which in the case of this project will be a text file with a string of characters representing the Y-chromosome.
- The mutate algorithm is a simple process that analyses a file and proceeds to change individual characters, in any text file, based on a probability provided by the user. This will mostly be used for analysing and comparing the behaviour of the copy model presented against other general-purpose compression tools: gzip, bzip2, tar, zstd, lz4 and 7zip.

With this project, we aim to learn more about compression algorithms, the benefits of using a probabilistic approach to compression and, possibly, the best practices in the field to better understand how a copy model functions.

Methodology

In preparation for this project, the goals and objectives were outlined to streamline the process of creating and testing the copy model or CPM. Through discussion, it was decided that the optimal path to proceed was to start by developing the “mutate” program as it required features that were to also be used by the CPM while also allowing the group to multitask by researching and testing general-purpose compression tools whilst the CPM was being worked on.

Mutator

Broadly speaking, the mutator reads a file and changes it based on a certain provided percentage, creating a new mutated file which will later be used for testing compression tools’ behaviour and the CPM’s prediction performance in comparison to said tools.

To begin, a UTF8 reader and parser were created, these will make a first read-through of the file to be analysed converting all symbols to their corresponding hexadecimal UTF-8 code. This way it will be possible to work with text containing non-ASCII characters such as the Portuguese language. Upon conversion of a symbol, the algorithm checks whether or not that symbol has already occurred and, in case it hasn’t, it adds it to the alphabet to use otherwise it skips it.

Following the above, a straightforward mutator was designed to change the original file’s content in a controlled fashion. The mutator takes in multiple flags, one for the input file, the name of the resulting mutated file, a buffer size and a probability as shown below:

```
./bin/mutate_program --inputFileName chry.txt --probability 20
                     --outputFileName mutated.txt --bufferSize
1024
```

The buffer size defines how big a buffer for analysing a file should be. This is used to be able to handle bigger files without having to store them in full in memory. The probability, provided by the user, is utilised to randomly change the symbols in the file to different symbols within the realm of the file’s alphabet.

Copy Model (CPM)

To understand what this CPM does, first, it’s necessary to understand what a CPM is. This is an algorithmic model that searches through a file trying to find redundancies (repetitions). By using a probability distribution it utilises the observed past occurrences to make predictions which if correct, imply that there is a pattern meaning that the data can be further compressed. For this, it takes in multiple parameters:

- an alpha, a smoothing parameter to avoid values of 0 in the probability calculations;
- size of the Kmer which is the size of the sequence being analysed;
- a threshold value for controlling when the model should turn itself off, resetting its anchors;
- And a buffer size which defines the size of the chunk of data to be analysed.

```
./bin/cpm_model --alpha 1 --k 8 --threshold 10
                --inputFileName chry.txt --bufSize 1024
```

The model created for this project makes use of 3 different buffers:

- a circular buffer for the Kmer which is represented as a data structure that can easily be updated with new elements without needing to resize the structure as it contains both head and tail pointers which allow for a continuous flow of data to be processed.
- Two buffers, one stays in the past with analysed symbols used as a cache to be accessed for reference once a sequence that's already occurred appears in the future in the second buffer.

Our model, reads a symbol at a time, taking into consideration the previous k symbols read, the Kmer. Then it tries to predict what the next symbol would be based on said Kmer. As stated above we make use of a double buffering structure to make predictions and store references. The data source is divided into chunks and each chunk is processed in the following manner:

1. The Kmer is read and the anchors hashmap is checked for repetitions.
2. If it did not occur in the past, its position is stored and in which of the two buffers it appeared.
3. If it already was in the hashmap, then the model checks in what buffer it happened and gets the symbol that came after the Kmer to make a prediction.
4. Upon making a prediction, the model gets its hit probability updated. If right the total bit count for the compressed file will be lower than if wrong.
5. Lastly, the model calculates the total estimated amount of bits needed to represent the data analysed.

It's important to note that, even though the data is divided into chunks, the copy model is still able to read symbols one at a time between both buffers and swaps the analysed chunk from buffer 2 to buffer 1 to analyse new data in buffer 2. On top of that, this model focuses on only finding predictions for the most recent occurrence of any sequence of symbols rather than a probability distribution for multiple occurrences. This will be further discussed in future work.

When estimating probabilities, our model uses the provided materials where Nh represents the number of hits, Nf the number of failed predictions and alpha is a smoothing parameter which prevents assigning 0 probability to events that haven't occurred. These are used with the following formula:

$$P(\text{hit}) \approx \frac{N_h + \alpha}{N_h + N_m + 2\alpha}$$

And the number of bits needed to represent each symbol is calculated by use of the formula:

$$-\log_2 P(s)$$

Discussion

Testing

The testing phase was executed using the provided Y-chromosome file(chry) and two additional text files: “The Great Gatsby” in its entirety and the first chapter of “Os Maias”. The text selections were intentional due to the fact that both of these books feature writing styles that could affect compression. “The Great Gatsby” is written in English with several words written in different languages (namely French), on the other hand, “Os Maias” contains wording that is convoluted as per the time when it was written while also, since it is in Portuguese, it contains plenty uses of UTF-8 characters which not all algorithms are prepared to handle.

As stated before, six widely used general-purpose compression tools were used for comparison and evaluation those being: tar, zstd, gzip, bzip2, 7zip and lz4. These tools served as benchmarks as well as, factors in seeing the different effects of different types of algorithms compressing the same files.

When using the above-mentioned tools, it was noted that some of them provided flags which allowed for changing the “compression strength” but those flags were kept as the default value. To find the copy model’s “default” values for the testing, multiple values of Alpha, Kmer size and Threshold were used.

Initially, the values were: k=1 and t=10. As k increased, past k=2, the compression amount became smaller meaning the estimated size of the compressed file was increasing. Following that, t was changed and it was found that the value of t did not affect performance whatsoever when it came to the chry file. Then, changes to alpha were made.

The ideal values found were:

- k=2
- t=N/A
- alpha=1

Results

Upon selecting the default values to use for the CPM’s parameters, it was put to the test against other compression algorithms. As can be seen in Table 1, the CPM performed very below average on the chry file however, when it came to the text files it performed seemingly much better than the compression tools.

At some stages, out of all the tested tools, tar performed the worst, even increasing file size past a 50% mutation rate and always maintaining the size of the chry file.

Name\Bytes	original	7zip	bzip2	gzip	lz4	tar	zstd	cpm
chry	22	4M	5.5M	6.0M	12M	22M	5.9M	17.4M
maias	52	21K	19K	24K	36K	60K	24K	19K
gatsby	272	89K	78K	104K	172K	280K	104K	103K
25%								
chry	22M	5.7M	6M	6.4M	13M	22M	6.7M	17.1M
maias	56K	36K	36K	36K	48K	60K	36K	20.6K
gatsby	284K	160K	164K	172K	260K	292K	172K	89.2K
50%								
chry	22M	5.8M	6M	6.4M	13M	22M	6.8M	17M
maias	56K	40K	40K	40K	56K	60K	40K	19.4K
gatsby	296K	192K	192K	200K	292K	300K	204K	87.5K
75%								
chry	22M	5.8M	6M	6.4M	13M	22M	6.8M	17M
maias	60K	40K	40K	44K	60K	60K	44K	22.9K
gatsby	304K	208K	204K	216K	304K	312K	220K	89.3K
100%								
chry	22M	5.9M	6M	6.4M	13M	22M	6.8M	17M
maias	60K	40K	40K	44K	60K	72K	44K	22.7K
gatsby	316K	212K	208K	224K	316K	320K	228K	99.6K

Table 1. Compression results of all tested tools

It's important however to note that there were issues regarding the results which ended up not entirely verified or fixed for that matter. The total amount of bits calculated leads to an ambiguous solution where it's not entirely definite whether it is or not the size of the compressed file or the number of bits that were compressed. This leads to further discussion of whether the copy model was performing better with a value of k = 2 or k = 17.

In the case of the model calculating the final file size, then the model would be providing astonishingly low numbers such as the chry file being compressed from 22MB to just under 300KB. However, what leads us to believe this isn't the reality is that, when trying to process the text files above 25% mutation, the values start being irrationally low even including 0 bits which is just not possible. The above results were all done using k=2 by removing the resulting "TotalBits" amount from the original files' sizes.

Additionally, the threshold parameter did not affect the result at all which, could be for a myriad of reasons such as the model never being wrong enough for it to even be considered or for example, the model being so wrong that it's almost always turning itself off and resetting.

Conclusions

In the discussion portion of this paper, it is to note that a lot of valuable insights were attained but also that many problems remain unsolved. The results are promising but require further work and exploration. Experimenting with generic compressors, provided us with interesting unexpected results such as the .tar files increasing in size depending on mutation percentage. This is especially interesting as tar is a very widely used compression tool namely in Linux-based systems. Overall this project contributes to the broader discourse on compression methods and algorithms, as well as, information theory.
In retrospect, though undeniably useful, the double buffering technique may have taken too much time away from other more important factors relating to the algorithm as the files to analyse were quite small and wouldn’t greatly affect any computers’ RAM.

Future Work

From here, ideally, we'd want to dedicate more time to improving the logging abilities of the program as it's still considerably difficult to understand what it is doing or has done. Efforts need to be put towards understanding the ambiguous nature of the total bit calculations and reevaluating the methodology implemented.
On top of that, new techniques should be experimented with as initially the algorithm was intended to read the Kmers from the final position of the sequence backwards however, we ended up doing the opposite though, there's no certainty in what benefits come from doing so.

References

"Advantages of using std::make_unique over new operator." *Stack Overflow*, 29 May 2016, https://stackoverflow.com/questions/37514509/advantages-of-using-stdmake-unique-over-new-operator.

Jain, Sandeep. "Iterators in C++ STL." *GeeksforGeeks*, 10 January 2023, https://www.geeksforgeeks.org/iterators-c-stl/.

Pratas, Diogo, et al. *Efficient compression of genomic sequences*. The number of genomic sequences is growing substantially. Besides discarding part of the data, the only efficient possibility for coping with this trend is data compression. We present an efficient compressor for genomic sequences, allowing both reference-. Aveiro, University of Aveiro.

Pratas, Diogo, et al. "GReEn: A tool for efficient compression of genome resequencing data." *ResearchGate*, IEETA/DETI University of Aveiro, 1 December 2011, https://www.researchgate.net/publication/51852079_GReEn_A_tool_for_efficient_compression_of_genome_resequencing_data.